Berkeley Data Analytics Stack (Beyond Spark & Shark)

Ion Stoica
UC Berkeley

What is Big Data used For?

Reports, e.g.,

- » Track business processes, transactions

Diagnosis, e.g.,

- » Why is user engagement dropping?
- » Why is the system slow?
- » Detect spam, worms, viruses, DDoS attacks

Decisions, e.g.,

- » Personalized medical treatment
- » Decide what feature to add to a product
- » Decide what ads to show

Data is only as useful as the decisions it enables

Data Processing Goals

Low latency (interactive) queries on historical data: enable faster decisions

» E.g., identify why a site is slow and fix it

Low latency queries on live data (streaming): enable decisions on real-time data

» E.g., detect & block worms in real-time (a worm may infect **1mil** hosts in **1.3sec**)

Sophisticated data processing: enable “better” decisions

» E.g., anomaly detection, trend analysis

One Reaction

Specialized models for some of these apps

» Google Pregel for graph processing

» Impala for interactive queries

» Iterative MapReduce

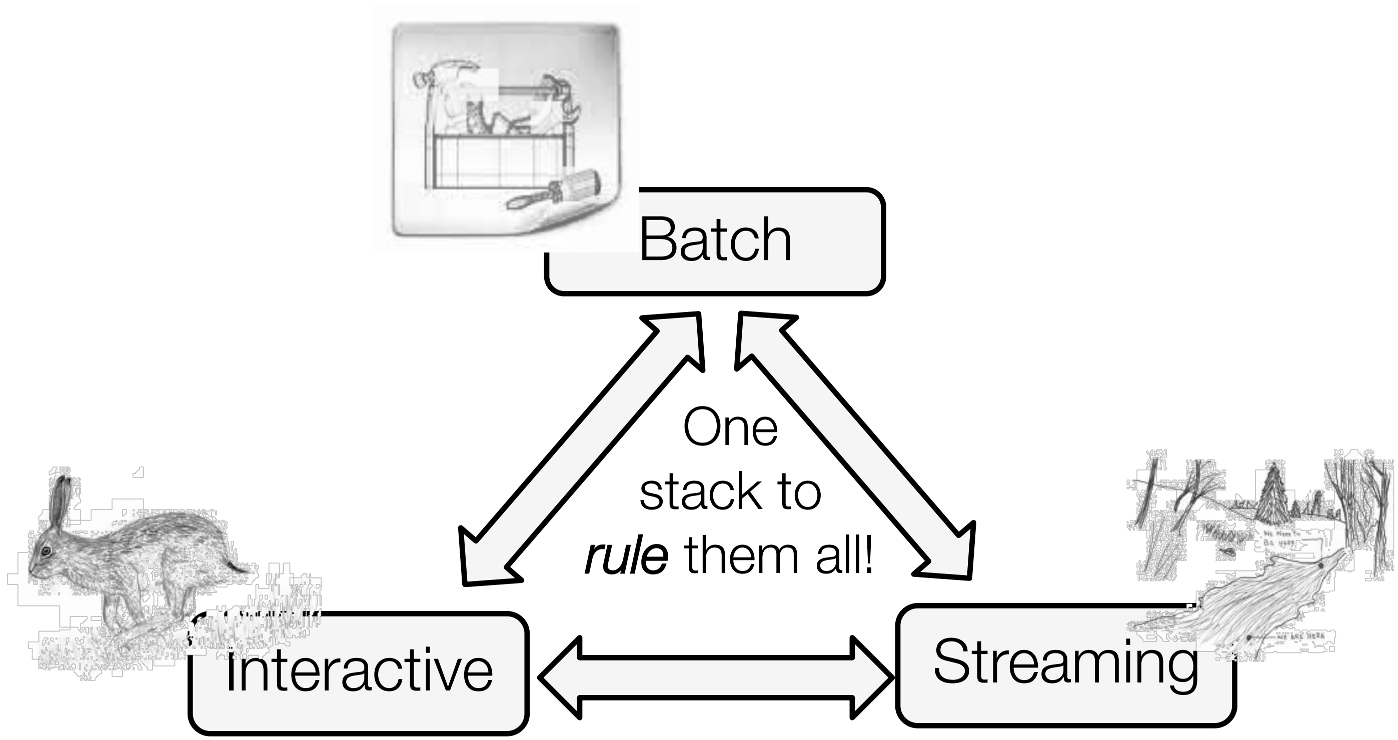
» Storm for streaming

Problem:

» Don’t cover all use cases

» How to *compose* in a single application?

Our Goals



Support *batch*, *streaming*, and *interactive* computations…

… and make it easy to compose them

Easy to develop *sophisticated* algorithms

Approach: Leverage Memory

Memory bus >> disk & SSDs

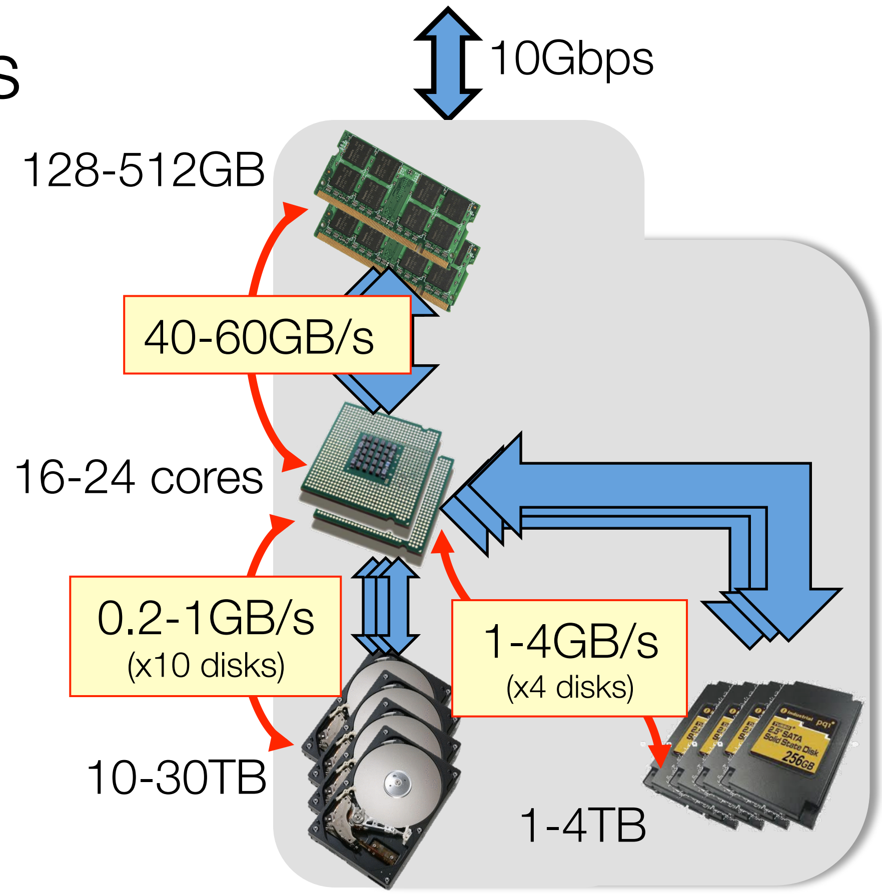
Many datasets fit into memory

» The inputs of over 90% of jobs in Facebook, Yahoo!, and Bing clusters fit into memory

» 1TB = 1 billion records @ 1 KB

Memory density (still) grows with Moore's law

» RAM/SSD hybrid memories at horizon



High-end datacenter node

Approach: Increase Parallelism

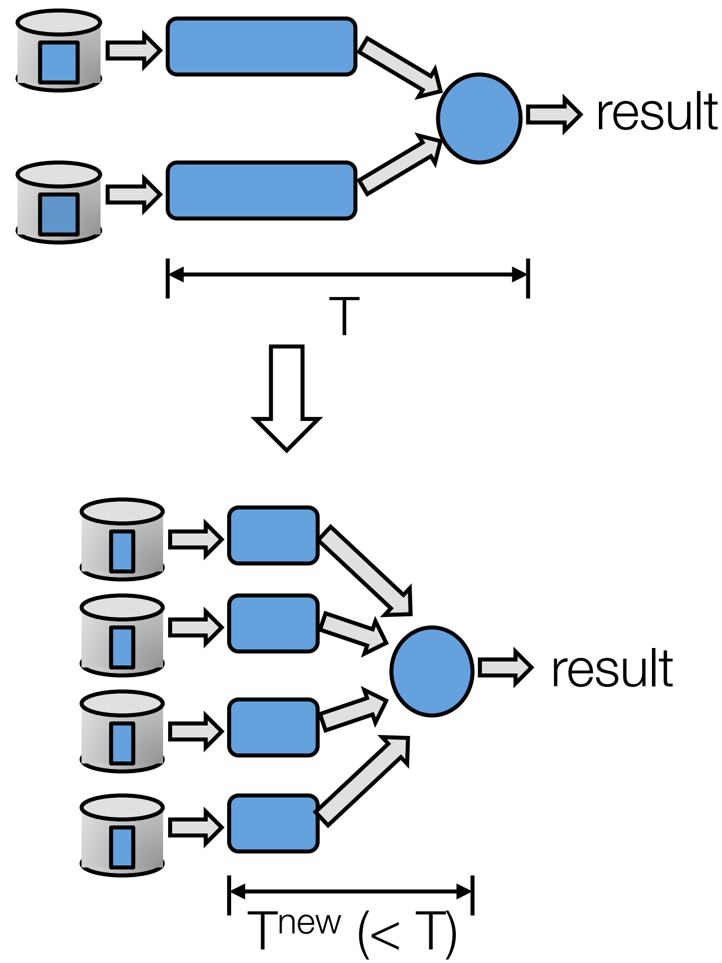
Reduce work per node → improves latency

Techniques:

» Low latency parallel scheduler that achieve high locality

» Efficient recovery from failures and straggler mitigation

» Optimized parallel communication patterns (e.g., shuffle, broadcast)



Spark: Interactive & Iterative Comp.

Achieve sub-second parallel job execution

Enable stages & jobs to share data efficiently

How?

» Resilient Distributed Datasets (RDDs): in-memory fault-tolerant storage abstraction
» Low latency scheduler
» Efficient communication patterns

Spark: Interactive & Iterative Comp.

Achieve sub-second parallel job execution

Enable stages & jobs to share data efficiently

How?

» Resilient Distributed Datasets (RDDs): in-memory fault-tolerant storage abstraction

» Low latency scheduler

» Efficient communication patterns

Resilient Distributed Datasets (RDDs)

How to ensure fault tolerance?

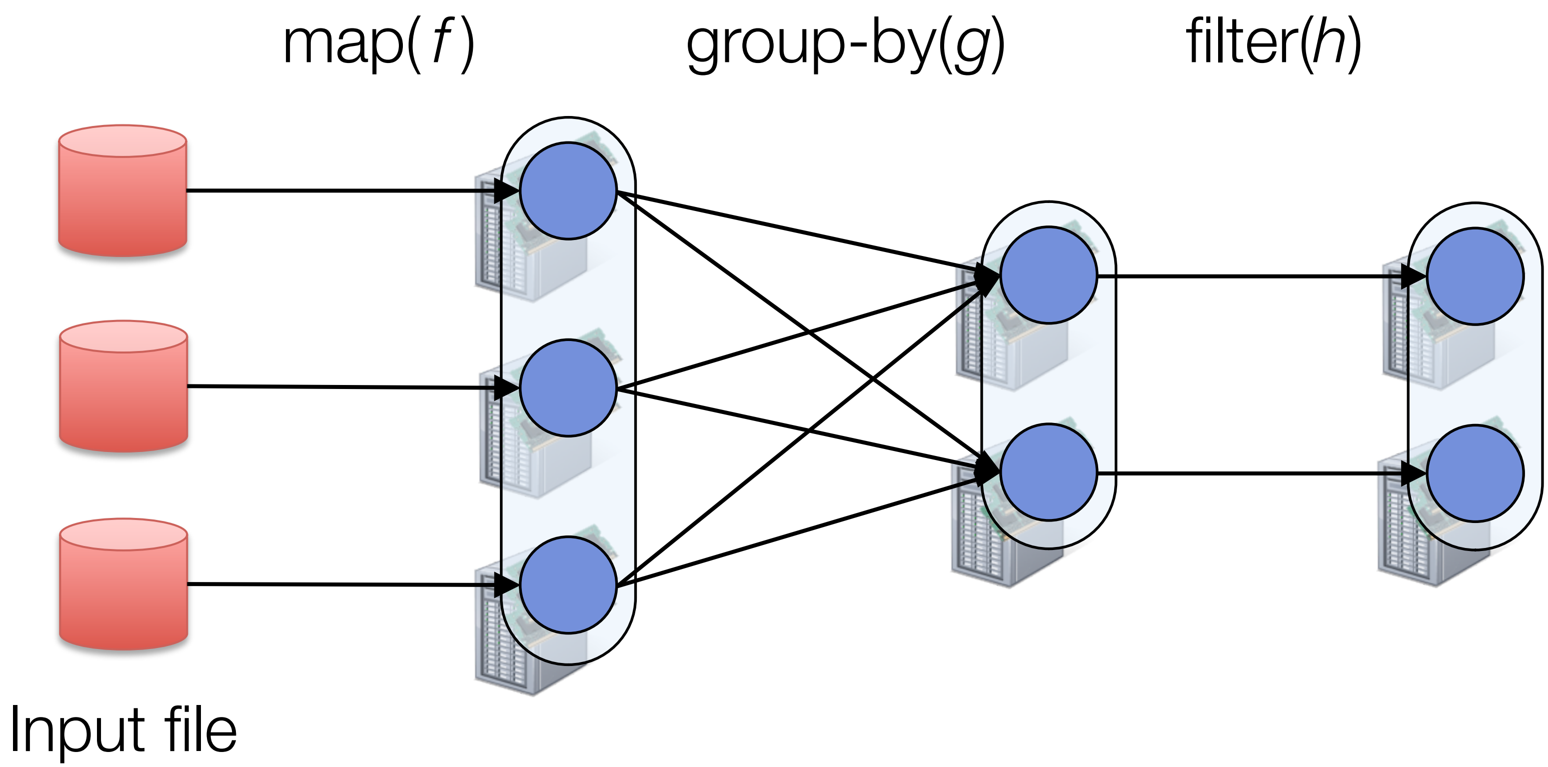
RDDs: restricted form of shared memory

- Immutable, partitioned sets of records
- Can only be built through *coarse-grained*, *deterministic* operations (map, filter, join, …)

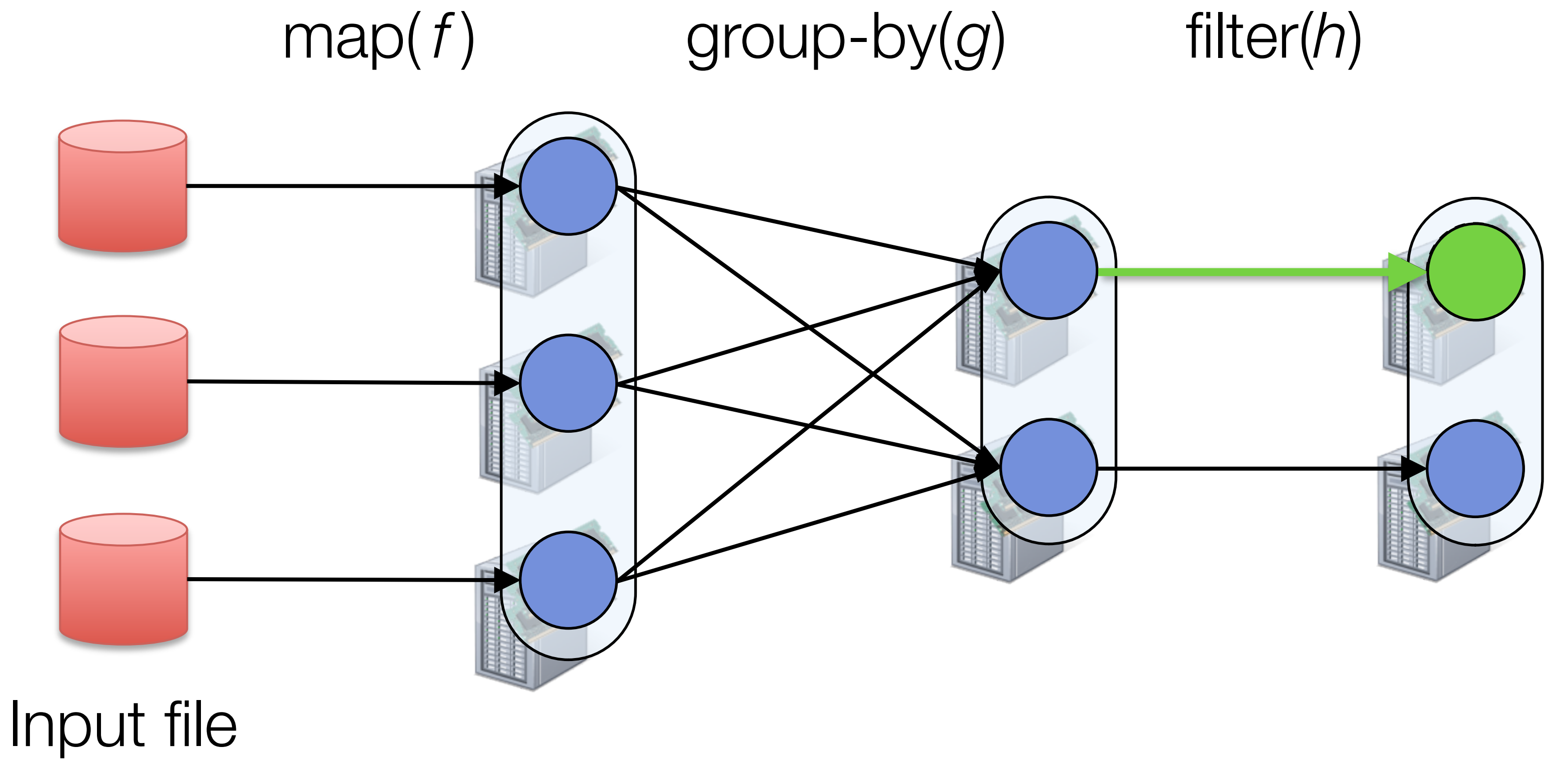
Use *lineage*

- Log one operation to apply to many elements
- Recompute any lost partitions on failure

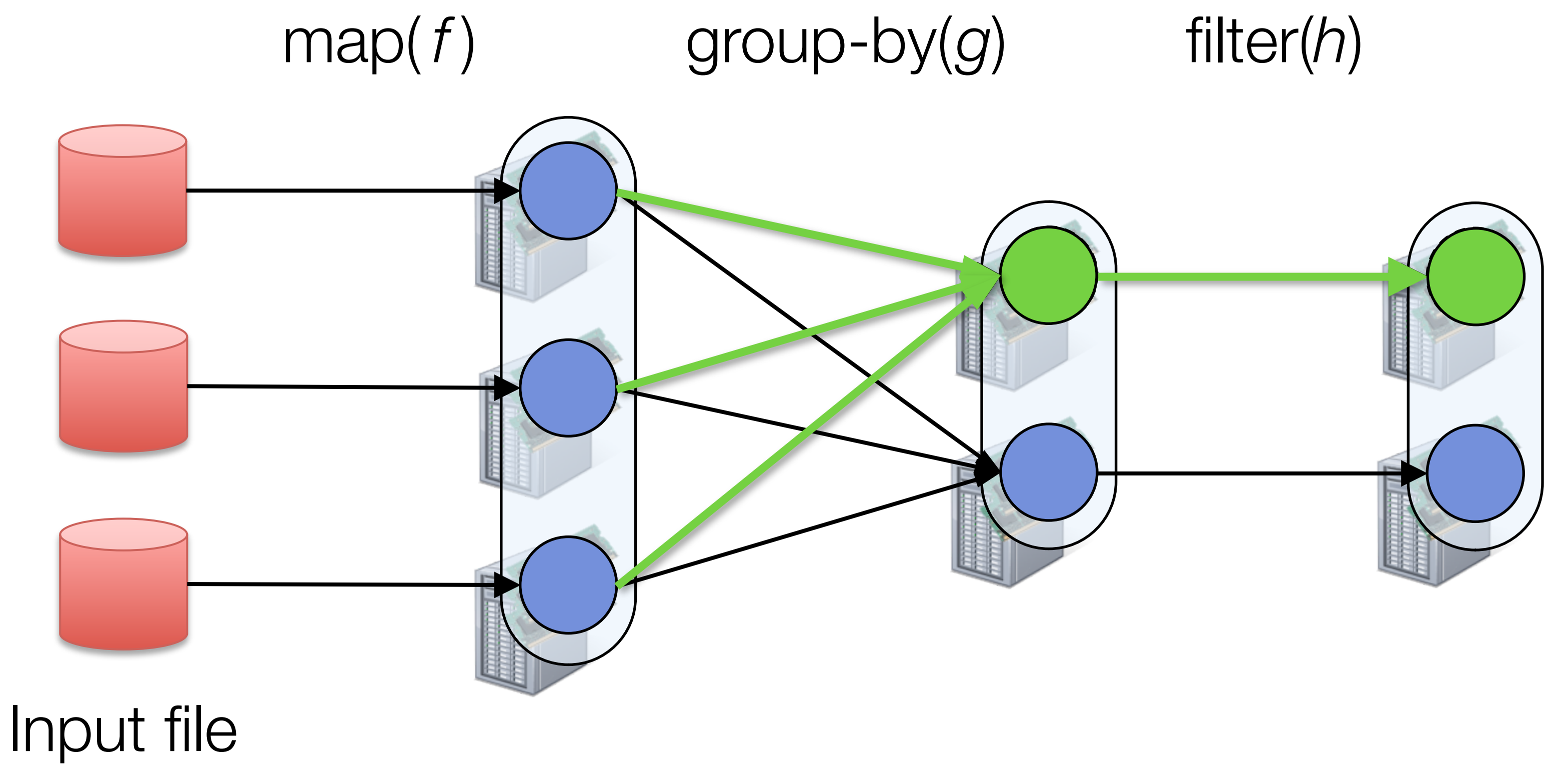
RDD Recovery



RDD Recovery



RDD Recovery



Generality of RDDs

Surprisingly, RDDs can express many parallel algorithms

» These naturally *apply the same operation to many items*

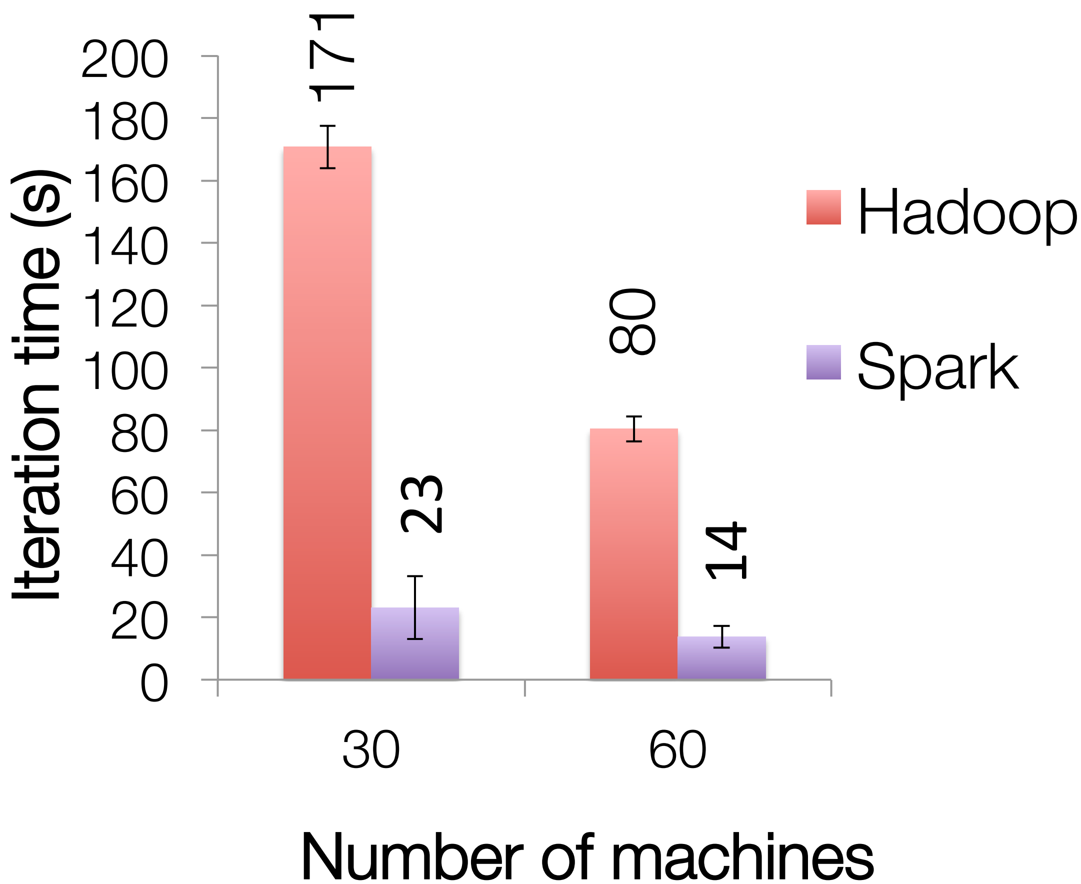
Unify many current programming models

» *Data flow models:* MapReduce, Dryad, SQL, …

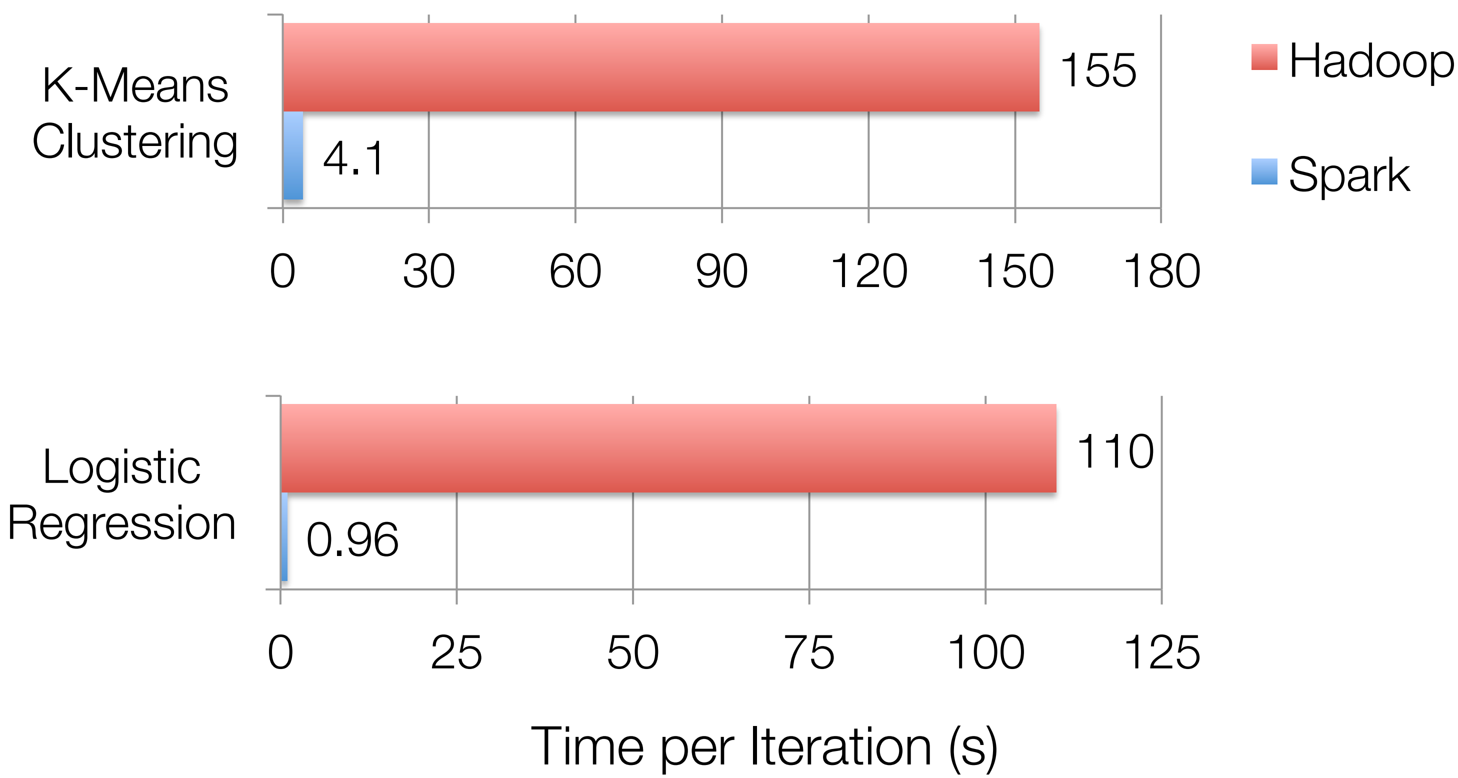
» *Specialized models* for iterative apps: Pregel, iterative MapReduce, GraphLab, …

Support new apps that these models don't

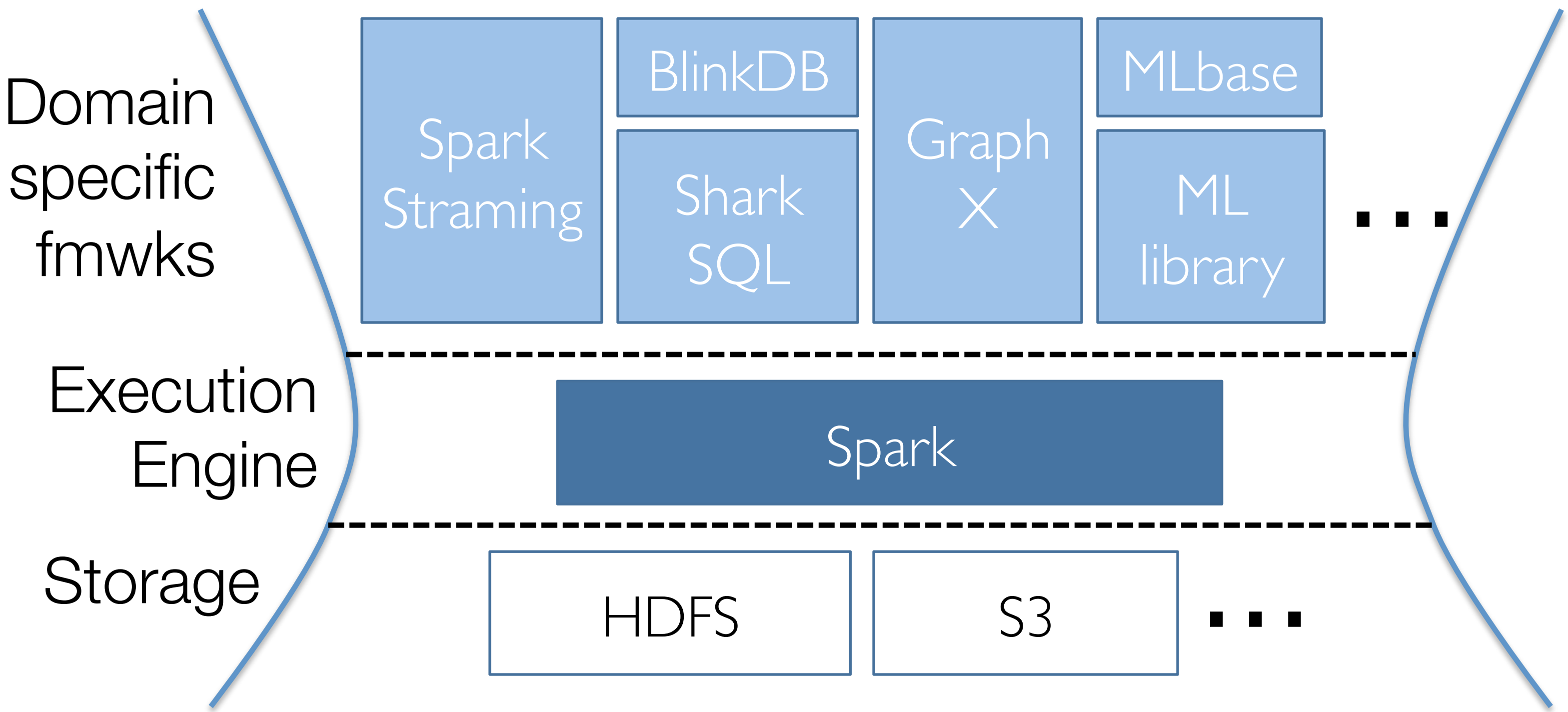
PageRank Performance



Other Iterative Algorithms

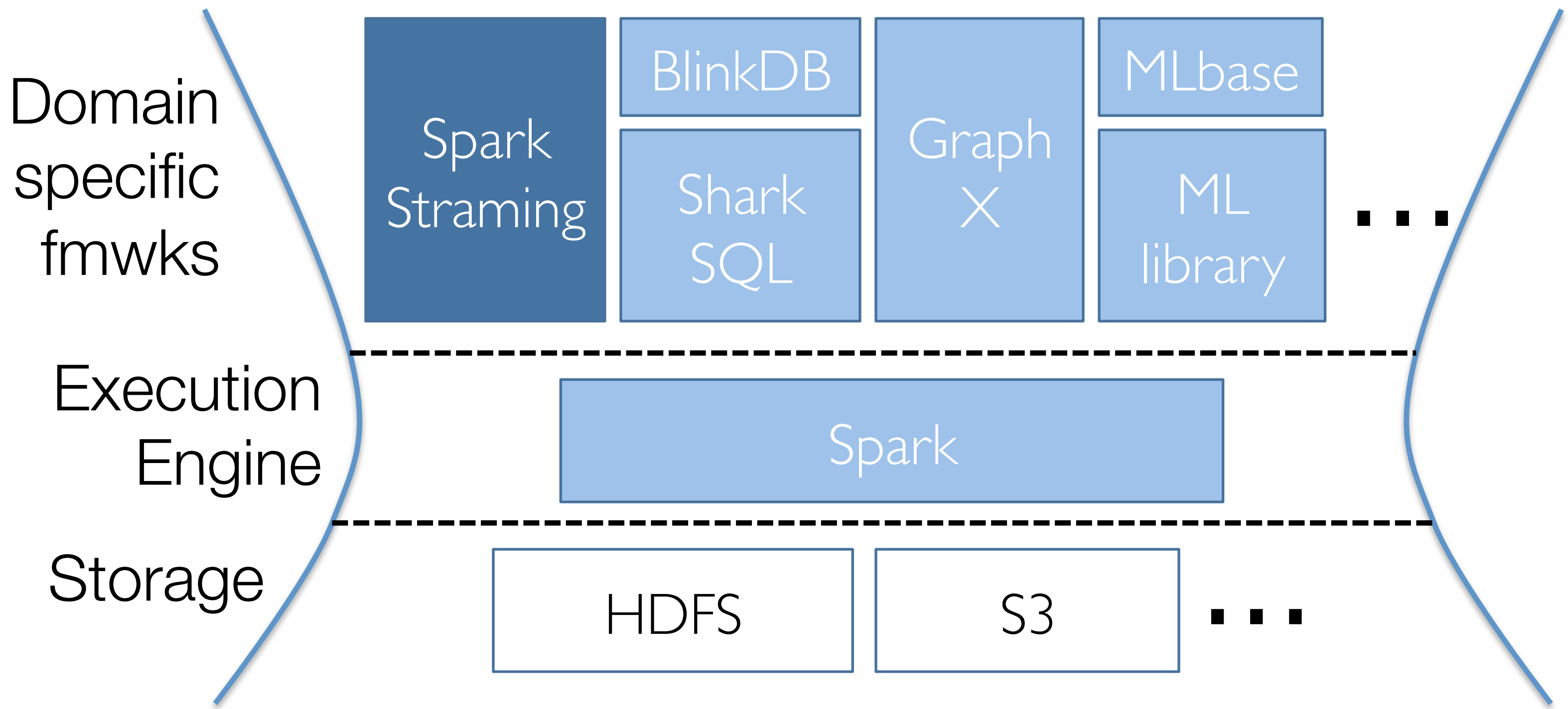


Spark: Narrow Waist of BDAS



Spark
Lightning-Fast Cluster Computing

Spark: Narrow Waist of BDAS



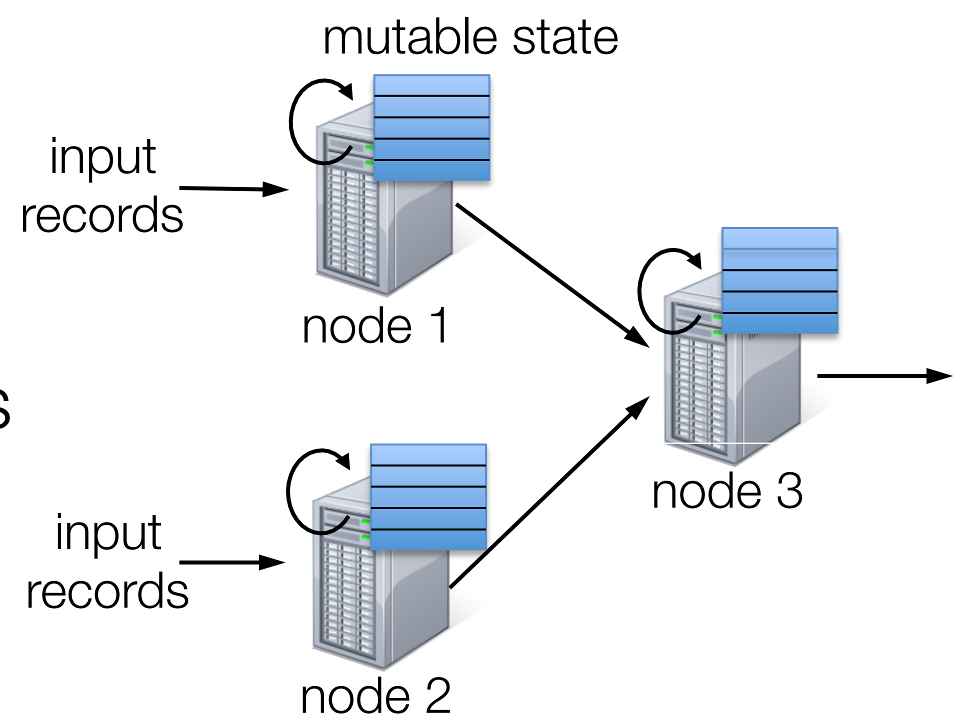
Existing Streaming Systems

Continuous processing model

- » Each node has long-lived state
- » For each record, update state & send new records

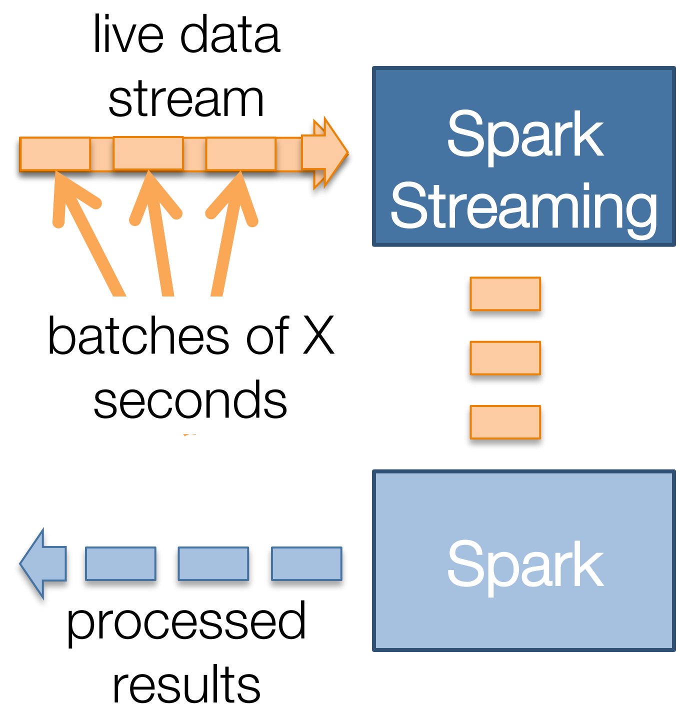
State is lost if node dies!

Making stateful stream processing fault-tolerant is challenging



Spark Streaming

Run a streaming computation as a **series of very small, deterministic batch jobs**



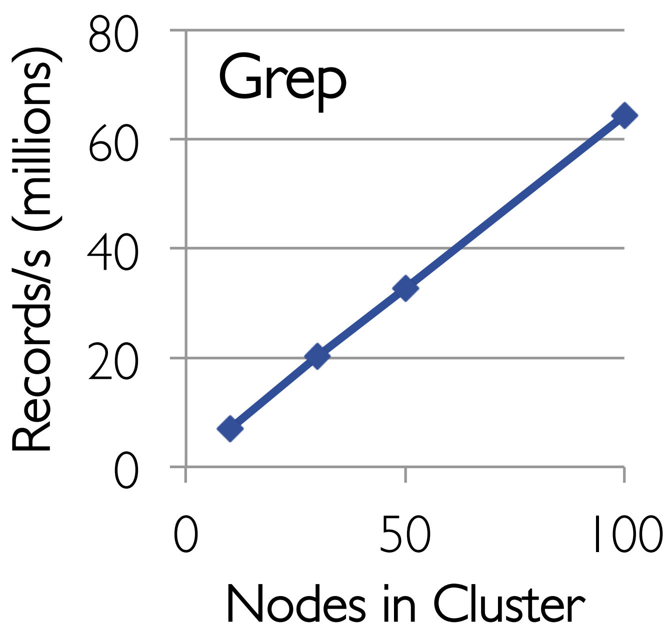
Divide live stream into batches of X seconds

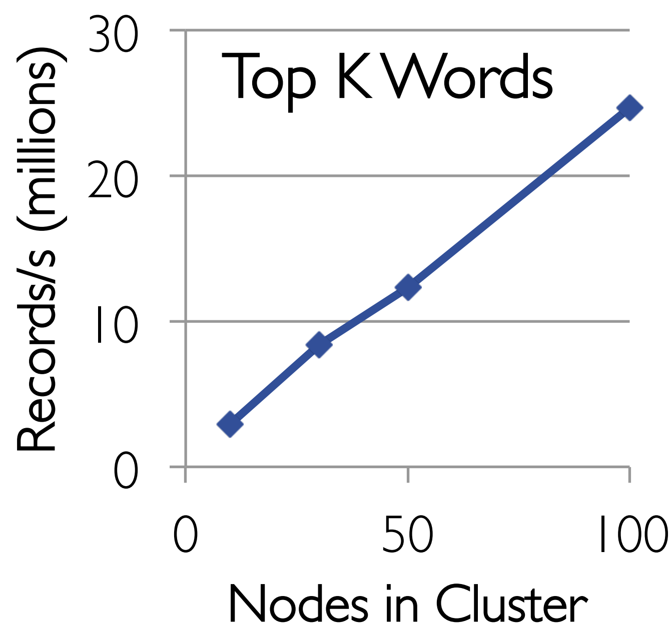
Spark treats each batch of data as RDDs

Return results in batches

How Fast Can It Go?

Can process over **60M records/s (6 GB/s)** on 100 nodes at **sub-second** latency



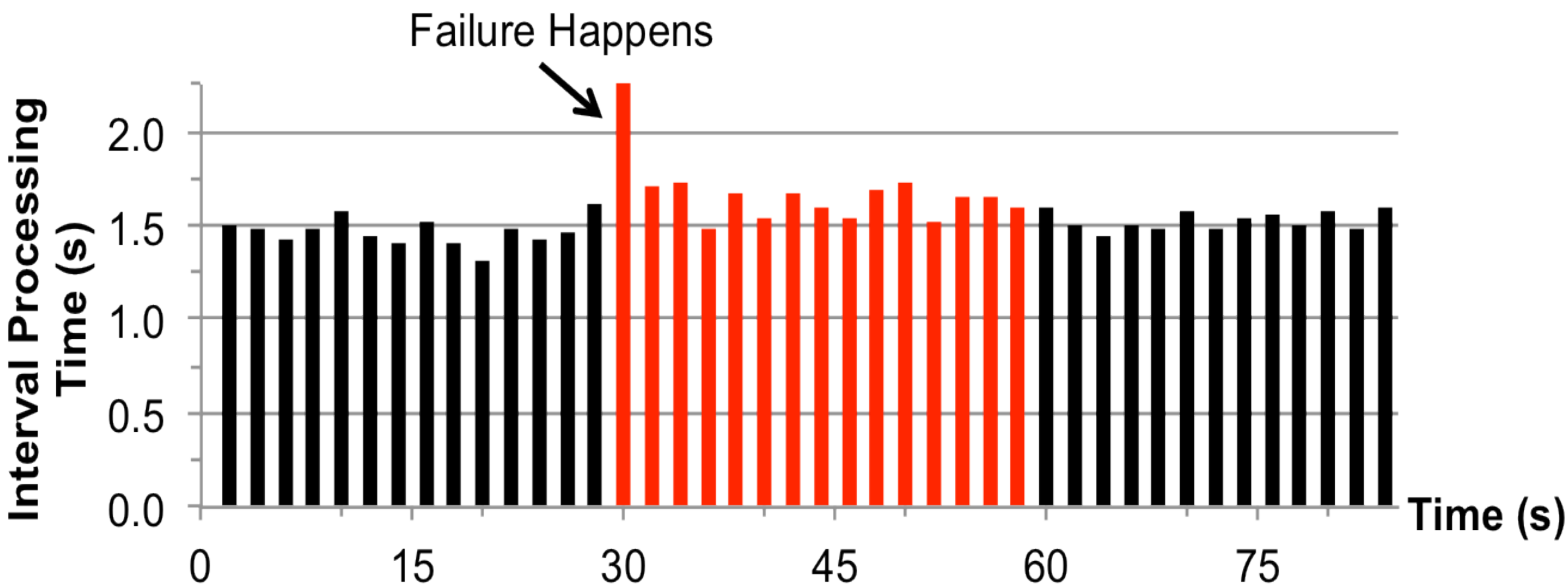


Maximum throughput for latency under 1 sec

How Fast Can It Recover?

Two second batches

Recovers from faults/stragglers within **1 second**

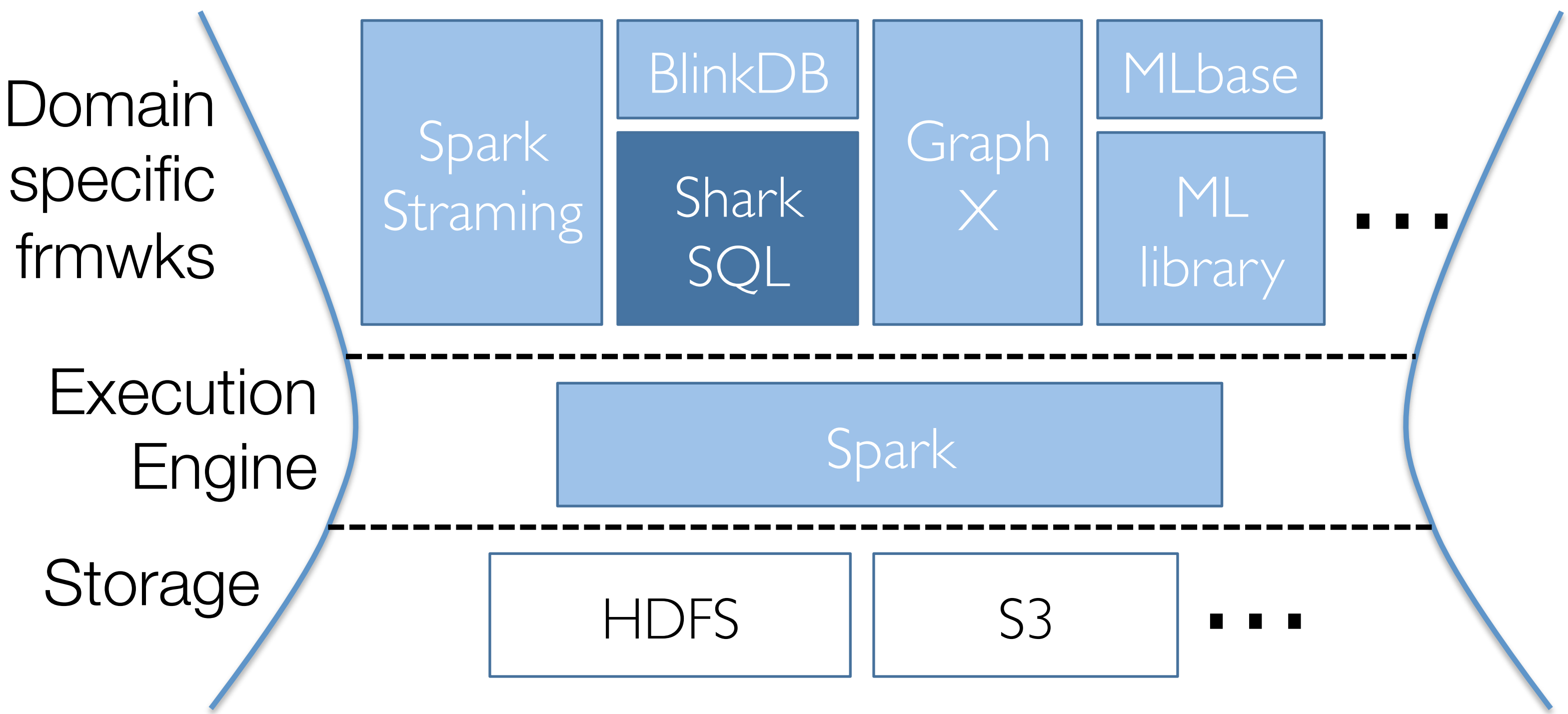


Sliding WordCount on 10 nodes with 30s checkpoint interval

Shark: Hive over Spark

Up to 100x faster when data in memory

Up to 5-10x faster even when data on disk



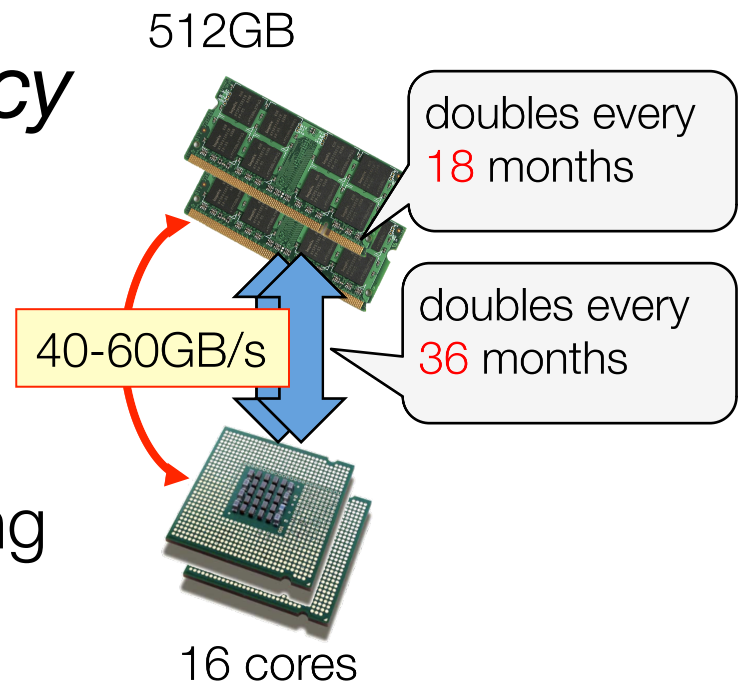
What Is Next?

Trade between result ***accuracy*** and ***response time***

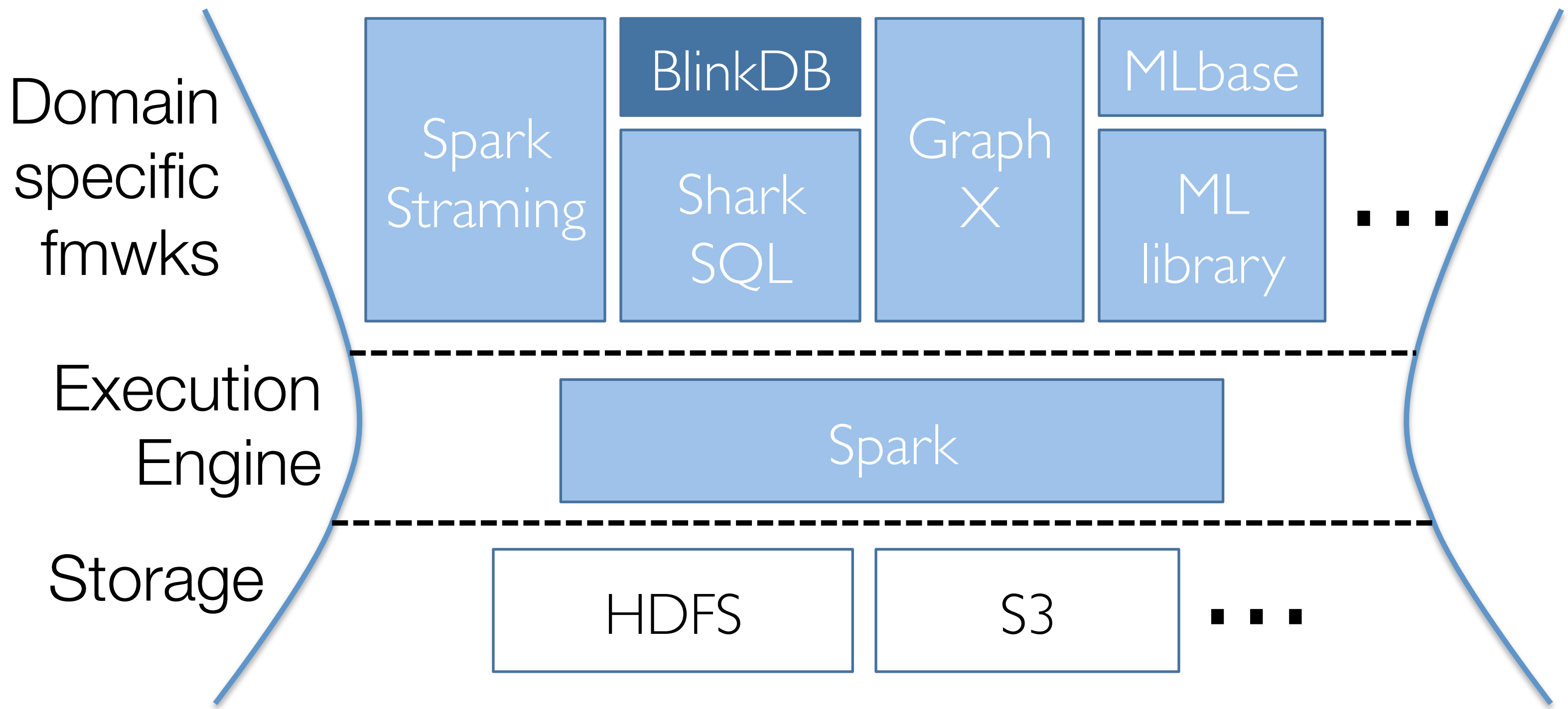
Why?

» In-memory processing doesn't guarantee interactive processing

- E.g., ~10's sec just to scan 512 GB RAM!
- Gap between memory capacity and transfer rate increasing



BlinkDB: Approximate Computations



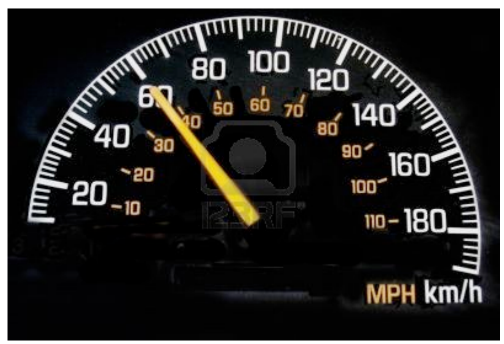
Key Insight

Don't always need *exact* answers

Input often *noisy*: exact computations do *not* guarantee exact answers

Error often acceptable if *small* and *bounded*

Best scale
± 0.5lb error



Speedometers
± 2.5 % error
(edmunds.com)

OmniPod Insulin Pump
± 0.96 % error
(www.ncbi.nlm.nih.gov/pubmed/22226273)

BlinkDB Challenges

How to estimate error bounds for arbitrary computations?

How do you know that technique you used is actually working?

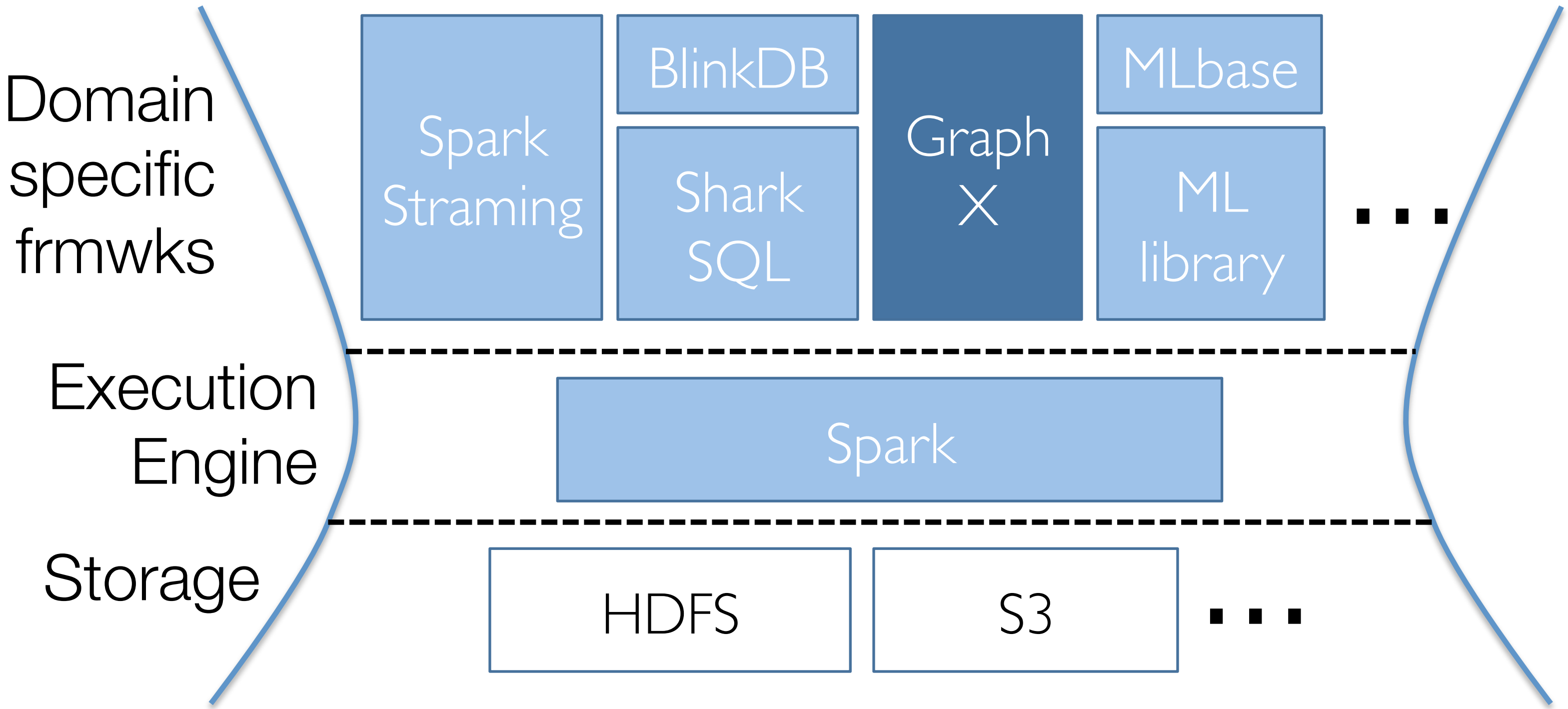
» Not trivial to check assumptions under which these estimates hold

» Many assumptions are sufficient, not necessary

What Is Next? Graph X

GraphLab API on top of Spark

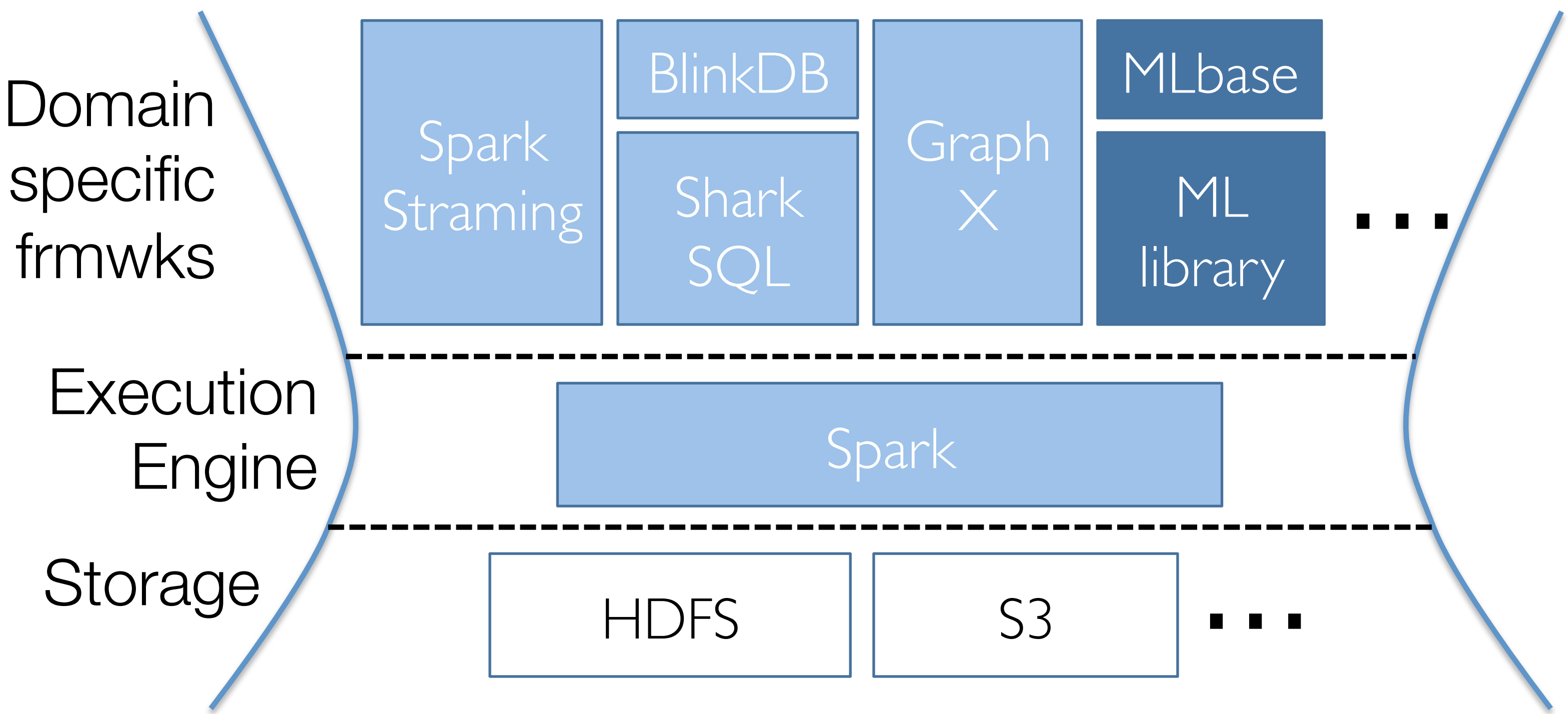
Leverage Spark's fault tolerance



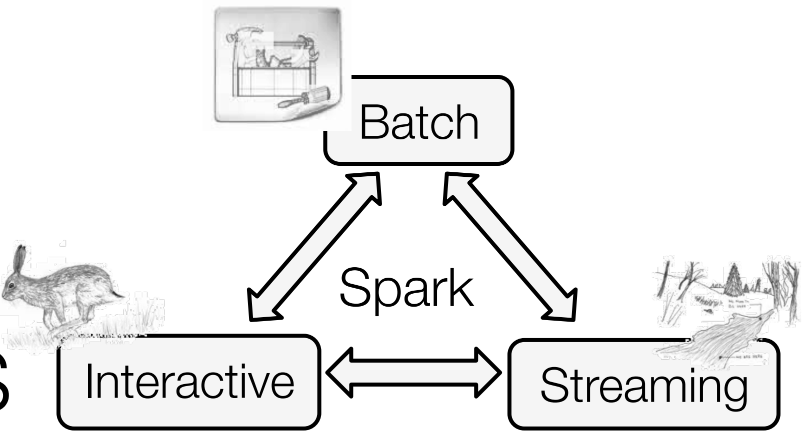
What Is Next? MLlib/MLbase

MLlib: Highly scalable ML library

MLbase: Declarative approach to ML



Summary



Spark: narrow waist of BDAS

» Unifies *batch*, *streaming*, and *interactive* comp.
» Ability to execute sub-second parallel jobs
» Enable job's stages and jobs to share in-memory data

Future work

» Sophisticated computations (Graph X, MLbase)
» Trade accuracy, speed, and cost (BlinkDB)

Vibrant open source community

» Used by tens of companies (e.g., Yahoo!, Intel, Twitter…)
» 60+ contributors from 17+ companies

amplab
UC BERKELEY